TITLE

SYSTEM AND METHOD FOR PERFORMING KERNEL-MODE OPERATIONS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to a system and method for performing kernel-mode operations, and in particular to a system and method for performing secure kernel-mode operations, executing a low-level debugging process.

Description of the Related Art

Using debugging processes, software engineers are the bridge between software and hardware, determining errors in software, such as BIOS (Basic Input/Output System), drivers, or operating systems, and in hardware.

Fig.1 is a schematic diagram showing architecture access system resources in a disk operating system (DOS). The architecture comprises application/tool 11, BIOS 13, authorized instruction 15, I/O (Input/Output) port 17 and memory 19. Traditionally, an operating system, such as disk operating system, opens all system resources to users, accessing BIOS 13 or executing low-level operations, performing authorized instruction 15 or accessing I/O port 17 and memory 19, through application/tool 11, none of which create serious problems during debugging.

Currently, operating systems must transit from the traditional CPU (Central Processing Unit) operating mode (real mode herein) to a 32-bit protected mode, providing enhanced operating efficiency and system resource

management. Under protected mode, the operating system restricts and prohibits most of system recourses, thus the accessibility of the system resources and operations are available to only those holding the highest authorization, such as Ring 0 authorization.

Fig. 2 is a schematic diagram showing kernel-mode operations performed through a kernel-mode driver in a Windows operating system, comprising application/tool 21, kernel-mode interface driver 23, BIOS 251, authorized instruction 253, I/O port 257 and memory 259. Application/tool 21 and kernel-mode interface driver 23 are executed in a user mode, while BIOS 251, authorized instruction 253, I/O port 257 and memory 259, are stored in a kernel mode. In conventional methods, to obtain Ring 0 authorization, kernel-mode interface driver 23 must be programmed using application/tool 21, via a driver call procedure, to enter kernel mode by performing a system call, directly accessing BIOS 251, or performing low-level operations such as executing authorized instructions 253 or accessing I/O port 257 and memory 259, implemented by driver development kit (DDK) in the Windows operating system.

Fig. 3 shows a schematic diagram performing kernel-mode operations using the driver development kit (DDK). Fundamentally, DDK packages desired data processed in a kernel mode to form an I/O request packet (IRP) 31, and informing a driver by a system call with a function DeviceIoControl() with respect to IRQ 31 and control codes. Next, hardware 37, such as authorized instruction 371, I/O port 373 and memory 375, is accessed by I/O

management system 33 through a series of transformations to hardware abstraction layer 35, and the procedure is complete.

Such a procedure, however, causes problems with system manufacturing, since, although normal use seeks to implement tasks with simple and intuitive kernel-mode operations, conditions become more complex and varied during system development, such that a specific kernel-mode driver may not be able to handle some situations. In addition, time limitations placed on some operations may affect accuracy. DDK can be difficult to work with, and applications proven in DOS may be difficult to transfer to other system architectures using a kernel-mode driver, requiring revision of software architecture or re-programming of source code.

SUMMARY OF THE INVENTION

Accordingly, an object of the present invention is to provide a system for performing kernel-mode operations, having applications with high (Ring 0) authorization to be performed in kernel mode.

Another object of the invention is to enable kernel-mode operations in a protected mode.

According to the objects described above, a kernel-mode interface generator generating a kernel-mode interface driver generating a call gate is provided, enabling the call gate to perform an operation with Ring 0 authorization in a kernel mode. Next, an authorization interface is provided to connect user and kernel modes, switching a process in the user mode to the kernel mode

through the call gate to perform an operation with the highest authorization.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention can be more fully understood by reading the subsequent detailed description and examples with references made to the accompanying drawings, wherein:

Fig.1 is a schematic diagram showing conventional system resource access in a disk operating system (DOS);

Fig. 2 is a schematic diagram showing conventional kernel-mode operations performed through a kernel-mode driver in Windows operating system;

Fig. 3 is a schematic diagram of conventional kernel-mode operations performed using a driver development kit;

Fig. 4 is a schematic diagram showing conventional processes in a user mode performing operations with the highest authorization in an operating system;

Fig. 5 is a schematic diagram showing the system for performing kernel-mode operations according to the present invention; and

Fig. 6 is a schematic diagram showing a process in a user mode switched to a kernel mode to perform operations with the highest authorization according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a system and method for performing kernel-mode operations in a protected mode.

Fig. 4 is a schematic diagram showing conventional processes in a user mode performing operations with the highest authorization in an operating system. Windows operating systems, such as Windows NT/2K/XP, are protected-mode operating systems, in which most system resources are restricted or accessible only with highest authorization. The highest authorization is required by a central processing unit (CPU) once it enters a kernel mode of an operating system, to access most system resources.

It is known that kernel mode for a process converts user-mode authorization to kernel-mode authorization by reasonable methods, as shown in Fig. 4.

Task switch 41 is a context switch operation generated by an operating system as a clock reading is obtained. Most contents of a CPU register for a current task are reserved as task switch operations occur, with the contents of the subsequent task loaded, task switch 41 switching tasks with different authorization levels.

System interrupt 43 is triggered by an operating system or system hardware, similar to the task switch, except that the CPU is not charged with contents of the register to be reserved by interrupt service routines (ISR). Interruptions usually occur with an

authorization-level switch, especially for interrupt requests (IRQ) by hardware or operating systems.

Authorization switch instruction (SYSENTER and SYSEXIT herein) 45 is provided by the CPU to enable a task to obtain the highest authorization through a specific entry point 451, initialized by operating systems and generally utilized by device drivers to instruct the operating system to assist in low-level hardware operations.

Call gate 47, having higher authorization than a general procedure call, is an authorization-level switch mechanism provided by the CPU for switching authorization, allocating a selector and an entry point 471 in a global descriptor table, enabling tasks corresponding to authorization level requests, to switch authorization through the entry point 471, and switching the authorization level back after tasks are complete.

Only authorization switch instruction 45 and call gate 47 are controllable for applications that determine when to switch authorization. However, when authorization switch instructions are used, related model-specific registers (MSR) must be set first, providing the single entry point. It is thus difficult to change or add any authorization entry points for applications. Therefore, the present invention uses the call gate to change or add authorization entry points by application.

Fig. 5 is a schematic diagram showing the system for performing kernel-mode operations according to the present invention. The architecture comprises a kernel-

mode interface generator (KMIf Generator) 51, a kernel-mode interface driver (IKM Driver) 53 and an authorization interface 55. Kernel-mode interface generator 51 generates kernel-mode interface driver 53 enabling correlation with authorization interface 55. Kernel-mode interface driver 53, dynamically generated by kernel-mode interface generator 51, generates call gate 531 and sets attributes of kernel variables used thereby. In addition, authorization interface 55 is the bridge between user and kernel modes and provides class methods enabling processes with user-mode authorization to perform kernel-mode operations.

Most authorized or restricted instructions, such as I/O port read/write, enable/disable interrupt, read/write MSR, and others, identified by CPU or operating system, are implemented by class methods. Furthermore, authorized instructions not supported by the present invention or system calls as transformations between linear addresses and physical addresses, necessary to pass authorization checking, provide callback functions like Ring0Function() for handling kernel-mode operations.

Briefly, authorization interface 55, according to the present invention, sends a system call to enable processes with lower authorization to acquire highest authorization through call gate 531 to access system resources or perform operations requiring higher authorization.

Fig. 6 is a schematic diagram showing a process in user mode switched to kernel mode to perform operations with the highest authorization according to the present

invention. Call gate is implemented by the following call far instruction:

call far CallGateSelector:CallGateOff

where "CallGateOff" is an arbitrary variable, a far pointer to call gate. As shown in Fig. 6, call gate selector 61 represented as a far pointer points to call gate descriptor 631 with selector information for entry point 65 and authorization level information for applicable call gates in global descriptor table 63. Accordingly, when a caller of call gate gives a call, it is determined whether the caller has corresponding authorization to determine whether the code-segment descriptor 633 to which the call gate points is taken. CPU performs stack switch if authorization checking is passed, with authorization level switch, and switches an instruction pointer to entry point 65, to obtain the address that sums up base address stored in code-segment descriptor 633 and offset stored in call gate descriptor 631.

The authorization level of the call gate is changed once its instruction pointer reaches the entry point, and the kernel-mode interface gives the call gate the highest authorization level enabling the caller to have the highest authorization to access system resources including I/O port read/write, memory access, and use of authorized instructions. The authorization level of the call gate is returned to user-mode authorization after it has performed the kernel-mode operations.

The method according to the present invention enables applications with user-mode authorization to

execute tasks with kernel-mode authorization and enables programming of kernel-mode applications with intuitive system operation aspects, shortening execution time for kernel-mode functions to achieve real-time performance optimization, and increase software flexibility in kernel-mode operations for dynamic program codes.

While the invention has been described by way of example and in terms of the preferred embodiments, it is to be understood that the invention is not limited to the disclosed embodiments. To the contrary, it is intended to cover various modifications and similar arrangements (as would be apparent to those skilled in the art). Therefore, the scope of the appended claims should be accorded the broadest interpretation so as to encompass all such modifications and similar arrangements.